arXiv:1508.00952v2 [math.OC] 25 Apr 2016

Graphical Newton

Akshay Srinivasan *
Dept. of Computer Science & Engineering,
University of Washington
Seattle, WA 98195

Emanuel Todorov
Dept. of Computer Science & Engineering,
University of Washington
Seattle, WA 98195

Abstract

Computing the Newton step for a generic function $f : \mathbb{R}^N \to \mathbb{R}$ takes $O(N^3)$ flops. In this paper, we explore avenues for reducing this bound, when the computational structure of f is known beforehand. It is shown that the Newton step can be computed in time, linear in the size of the computational-graph, and cubic in its tree-width.

1 Introduction

Newton's method forms the basis for many second-order methods in Nonlinear-optimization; it is also the core technique used in Interior point methods. It's applicability to large-scale programming, however, is often limited due to the run-time complexity in computing the Newton step.

For a generic function $f : \mathbb{R}^N \to \mathbb{R}$, computing the Hessian requires atleast $O(N^2)$ flops; further inverting the matrix requires $O(N^\gamma)$ flops ($\gamma = 3$, in practice). This is computationally infeasible for many problems in practice.

Often, however, one is also given access to the the *computational structure* of the objective. The computer routine for calculating the objective $f(\cdot)$ can be represented as a Directed Acyclic Graph [DAG] mapping inputs to $f(\cdot)$ *via* intermediary nodes.

For instance, the objective function for the canonical optimal-control problem is given by,

$$\min_{u_0,u_1,\ldots,u_{n-1}} \left[\jmath(u_0,\ldots,u_n) \triangleq \sum_{i=0}^{n-1} \ell_i(x_i,u_i) + \ell_n(x_n) \right], \quad \forall i, x_{i+1} \leftarrow \mathtt{f}(x_i,u_i), \tag{1}$$

where the dynamics and local-objectives of the system are given by $\mathtt{f}(\cdot,\cdot)$, and $\ell_i(\cdot,\cdot)$ respectively. The infix operator '$\leftarrow$' indicates that the value appearing on the right-hand side, is given the placeholder symbol present to its left; we explicitly distinguish this from the '$=$' operator, which is taken to represent a constraint.

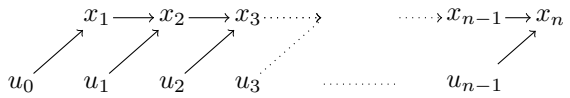

Figure 1: Optimal control problem: The dynamical system states are represented by $\{x_i\}$, and the control by nodes $\{u_i\}$.

The order of computation for the objective (1) can be represented by a linear-chain (Figure 1). Lacking constraints, the apparent sparsity in (1), is entirely destroyed once all the placeholders are substituted for,

$$\begin{aligned}\jmath(u_0,\ldots,u_n) =& \ell_0(x_0,u_0) + \\ & \ell_1(\mathtt{f}(x_0,u_0),u_1) + \\ & \ell_2(\mathtt{f}(\mathtt{f}(x_0,u_0),u_1),u_2) + \ldots.\end{aligned}$$

The Hessian of $\jmath(\cdot)$ thus being dense, implies a run-time that is cubic in the input dimensions for the Newton step computation; computing the Hessian itself is quadratic.

By contrast, once the problem (1) is written in its constrained form (by replacing '$\leftarrow$' with '$=$'), the sparsity of the resulting Karush-Kuhn-Tucker [KKT] system, readily allows for computing the SQP/Lagrange-Newton step in linear time [19]. Such a transformation, however, comes at the cost of increasing the size of the optimization problem, abandoning state feasibility, and increased implementation complexity.

The question which this paper answers, is whether there exist *general* techniques, which allow exploiting the sparsity of the problem, while working solely with the input variables. Note that these are not questions merely about elimination orders, but are also verily algebraic in nature.

AUTOMATIC DIFFERENTIATION: Research on Automatic Differentiation [AD] has produced many techniques for exploiting the *computational structure* of generic functions.

*Email:{akshays,todorov}@cs.washington.edu

They are routinely employed for efficient calculation of gradients and Hessian vector products [9]. The applicability of AD to second-order optimization is, however, quite limited.

AD is typically used either for computing the entire Hessian matrix, or for calculating Hessian vector products for use in Nonlinear Conjugate Gradient Descent [CG]. Hessians are computed, by accumulating one column at a time *via* calls to the Hessian vector product routine [9]. The sparsity of the Hessian can be exploited in reducing the number of such calls [3], but structured problems such as (1) will not allow for any such economy. Compositional chains of functions, such as those in optimal control (1) (Figure 1), not only serve to make Hessians dense but can also lead to condition numbers, exponential in their diameters. Large condition numbers are likely to negate any computational advantages offered by methods like CG.

The above techniques form the Hessian matrix, directly or indirectly, before computing the Newton step. This stands in contrast with the root-finding problem, for which there do exist methods for directly computing the Newton-Raphson step [17] [9] [6]. The root finding problem involves the inversion of the Jacobian of a function - rather than the Hessian - and these methods reduce this computation to that of inverting a sparse matrix [17]. The Newton step (for optimization) can also be computed using this method by formulating it as a root-finding problem on the gradient. This, however, results in non-symmetric matrices depending on the computational graph of the gradient, as opposed to the function itself. The latter graph is transitively closed, and hence, analyses for the above Newton-Raphson AD algorithm apply to the gradient, and are difficult to extend to the underlying objective [6].

DYNAMIC PROGRAMMING: The question posed earlier, has already been answered in the affirmitive, for the optimal control problem. There exists an algorithm for optimal control, based on Dynamic Programming, that exploits algebraic dependencies in (1), in order to compute the Newton-step in only linear time [5] [10] [15].

The run-time of this algorithm is the direct result of the sparsity of the corresponding constrained problem [5] [19][13]. The band-structure of the relevant KKT system allows for solving the system in linear time [19]. The relationship between computing the Newton-step (Hessian of the objective), and computing the Lagrange-Newton step (Hessian of the Lagrangian), is established by noting that there exist multiplier values such that both compute the *same* result [5].

Such algorithms are routinely employed by practitioners for updating *control policies* in real-time, while maintaining a feasible trajectory. These algorithms have been extended to Extended Kalman Filtering (EKF), as well as various other formulations of the control problem [16] [14] [1] [20].

OVERVIEW: We generalize such algorithms, by using Hessian vector product equations from AD, to relate the computation of Newton step and Lagrange-Newton step, for arbitrary structured objectives.

We then extend this framework to structured optimization problems with equality constraints.

Further, we show that solving the resultant KKT systems can be accomplished in time $\tilde{O}(\mathrm{tw}^3)$, where 'tw' is the tree-width of the canonical computational graph.

Finally, we show results from numerical experiments.

2 Notation

Let $\mathcal{G}$ be a Directed Acyclic Graph [DAG], and let each vertex $v \in V[\mathcal{G}]$, be associated with *state* $\mathtt{S}_v \in U_v \subset \mathbb{R}^{n_v}$, taking values in an open set. Denote by $\delta^+(v)$, the parents of $v \in V[\mathcal{G}]$, and by $\delta^-(v)$ its children; let $\mathtt{S}_A$ be the (labelled) concatenation of *state*s, associated with vertices in set $A \subset V[\mathcal{G}]$. Define the set of *input* nodes $X = \{x_1, x_2, \ldots, x_n\} \triangleq \{v \mid \delta^+(v) = \emptyset, v \in V[\mathcal{G}]\}$, to be the parentless vertices of $\mathcal{G}$.

An objective function $f : U_{x_1} \times \ldots U_{x_n} \to \mathbb{R}$, has the *computational structure* given by the tuple $(\mathcal{G}, \{\varphi_v\}, \{\ell_v\})$, if it can be written as the sum of local objectives $\ell_v : \prod_{z \in \{v\} \cup \delta^+(v)} U_z \to \mathbb{R}$, on the graph $\mathcal{G}$,

$$\begin{aligned} f : (\mathtt{S}_{x_1}, \ldots, \mathtt{S}_{x_n}) &\mapsto \sum_{v \in V[\mathcal{G}]} \ell_v(\mathtt{S}_{v \cup \delta^+(v)}), \\ \mathtt{S}_v &\leftarrow \varphi_v(\mathtt{S}_{\delta^+(v)}), \quad \forall v \in V[\mathcal{G}], \delta^+(v) \neq \emptyset. \end{aligned} \tag{2}$$

The *state* of a non-input node $v \in V[\mathcal{G}]$ in (2), is defined recursively as $\mathtt{S}_v \leftarrow \varphi_v(\mathtt{S}_{\delta^+(v)})$, for some given function $\varphi_v : \prod_{z \in \delta^+(v)} U_z \to U_v$. It follows since $\mathcal{G}$ is a DAG, that $\mathtt{S}_{V[\mathcal{G}]}$ and hence $f(\cdot)$, is uniquely determined from the input $\mathtt{S}_X$, and functions $\{\varphi_v\}$. The order of computation for the objective is given by the topological ordering of $\mathcal{G}$, and the DAG $\mathcal{G}$ is called the *computational graph* of $f(\cdot)$. The computer routine for calculating any objective function, can be represented by such a structure [9].

In the following sections, the symbolism $\partial_u v$ is used as a shorthand for $\frac{\partial \mathtt{S}_v}{\partial \mathtt{S}_u}\big|_{\mathtt{S}_X}$. The derivatives of functions with respect to $\mathtt{S}_u$ are similarly denoted by the operator ∂_u; that with respect to a (labelled) set $A = \{v_1, v_2, \ldots\} \subset V[\mathcal{G}]$ by $\partial_A \triangleq [\partial_{a_1}, \partial_{a_2}, \ldots]$.

3 Newton step

Consider the objective function in (2), defined by the tuple $(\mathcal{G}, \{\varphi_v\}, \{\ell_v\})$. The optimization problem of interest is the

following,

$$\min_{\mathtt{S}_{x_1},\ldots,\mathtt{S}_{x_n}} \left(f \triangleq \sum_{v\in V[\mathcal{G}]} \ell_v(\mathtt{S}_{v\cup\delta^+(v)}) \right), \quad (3)$$
$$\mathtt{S}_v \leftarrow \varphi_v(\mathtt{S}_{\delta^+(v)}), \quad \forall v \in V[\mathcal{G}], \delta^+(v) \neq \emptyset,$$

and the corresponding constrained problem is obtained by replacing the operator '$\leftarrow$' by '$=$' in (3).

In the following, we consider first the constrained formulation of (3), and define the KKT system involved in computing the Lagrange-Newton step; we then relate these to computing the Newton step.

3.1 Lagrange-Newton

The Lagrangian for the constrained form of (3), is given by,

$$\mathcal{L}(\mathtt{S}_{V[\mathcal{G}]}, \lambda) \triangleq \sum_{v\in V[\mathcal{G}]} \ell_v(\mathtt{S}_{v\cup\delta^+(v)}) + \sum_{\substack{v\in V[\mathcal{G}],\\ \delta^+(v)\neq\emptyset}} \lambda_v^{\mathtt{T}} h_v(\mathtt{S}_{v\cup\delta^+(v)}),$$

where,

$$\forall v \in V[\mathcal{G}], \delta^+(v) \neq \emptyset, \quad h_v(\mathtt{S}_{v\cup\delta^+(v)}) \triangleq \varphi_v(\mathtt{S}_{\delta^+(v)}) - \mathtt{S}_v, \quad (4)$$

and the vector λ is the labelled concatenation of all λ_v's.

The necessary first order conditions for optimality of this problem are given by [12],

$$\partial_V \mathcal{L}(\mathtt{S}_V^*, \lambda^*) = 0, \quad h(\mathtt{S}_V^*) = 0. \quad (5)$$

The Lagrange-Newton step for solving this system of equations, around a nominal $(\tilde{\mathtt{S}_V}, \lambda)$, entails solving the following KKT system [12],

$$\begin{bmatrix} \partial_V^2 \mathcal{L} & \partial_V h^{\mathtt{T}} \\ \partial_V h & 0 \end{bmatrix} \begin{bmatrix} \delta\mathtt{S}_V \\ \delta\lambda \end{bmatrix} = \begin{bmatrix} -\partial_V \mathcal{L} \\ -h \end{bmatrix}. \quad (6)$$

Sequential Quadratic Programming [SQP], involves taking a step along $(\delta\mathtt{S}_V, \delta\lambda)$ and iteratively solving for the first order conditions. In the following section, it will be shown that there exist values for Lagrange multipliers, depending only on the inputs, such that the solution to (6), yields the Newton step for the unconstrained objective.

3.2 Unconstrained Newton

We recollect certain defintions from AD, and then continue to present one of the central results of the paper.

REVERSE AD: The first derivatives of the objective $f(\cdot)$ can be calculated by applying the chain rule over $\mathcal{G}$,

$$\forall v, \quad \partial_v f = \sum_{s\in v\cup\delta^-(v)} \partial_v \ell_s + \sum_{d\in\delta^-(v)} \partial_d f^{\mathtt{T}}\, \partial_v d; \quad v \in \delta^+(d) \Rightarrow \partial_v d \triangleq \frac{\partial \varphi_d(\mathtt{S}_{\delta^+(d)})}{\partial \mathtt{S}_v}. \quad (7)$$

Since $\mathcal{G}$ is a DAG, there exist child-less nodes (*i.e* $\delta^-(v) = \emptyset$), from which the above recursion can be initialized. The recursion then proceeds backward in the depth first search order on $\mathcal{G}$. This algorithm is known as reverse-mode AD [9].

HESSIAN VECTOR AD: A change in the inputs $\delta\mathtt{S}_X$, results in the first-order change in the derivative, $\delta[\partial_v f] \triangleq \partial^2_{Xv} f \cdot \delta\mathtt{S}_X$, which is given by the Hessian vector product. Computing the Newton step is thus, equivalent to finding a $\delta\mathtt{S}_X$ such that, $\delta[\partial_X f] = -\partial_X f$.

Applying chain-rule over the DAG $\mathcal{G}$, for all terms in (7), we obtain,

$$\begin{aligned} &\forall v, \quad \delta[\partial_v f] = \\ &\quad \sum_{s\in v\cup\delta^-(v)} \left(\sum_{a\in v\cup\delta^+(s)} \partial^2_{av}\ell_s \cdot \delta\mathtt{S}_a \right) + \\ &\quad \sum_{d\in\delta^-(v)} \left(\delta[\partial_d f]^{\mathtt{T}} \partial_v d + \sum_{a\in\delta^+(d)} (\partial_d f^{\mathtt{T}}\, \partial^2_{av} d) \cdot \delta\mathtt{S}_a \right); \\ &\forall a, \quad \delta\mathtt{S}_a = \sum_{d\in\delta^+(a)} \partial_d a \cdot \delta\mathtt{S}_d. \end{aligned} \quad (8)$$

These equations can be solved, for a given $\delta\mathtt{S}_X$, by a forward-backward recursion similar to the one used for solving (7) [9]. Computing the Hessian-vector product in this manner takes time $\tilde{O}(\omega(\hat{\mathcal{G}})^2)$ [1] [9], where $\omega(\hat{\mathcal{G}})$ is the clique number of the moralization of $\mathcal{G}$.

NEWTON STEP: The problem of interest is, however, the exact inverse: find a $\delta\mathtt{S}_X$, such that $\delta[\partial_X f] = -\partial_X f$. This question is answered by the following theorem.

Theorem 1 *(Newton step) The Newton step for the objective (2) is given by the Lagrange-Newton step (6), when $\mathtt{S}_V$ is feasible and when $\forall v, \lambda_v = \partial_v f$ as defined in (7).*

Proof. The second equation in (8) is equivalent to $\partial_V h \cdot \delta S_V = -h$, in (6). Rearranging the first equation from (8), and setting $\delta[\partial_v f] = -\partial_v f$ for all inputs, we obtain $\forall v$,

$$\begin{aligned} 0 = &\sum_{\substack{s\in v\cup\delta^-(v),\\ a\in v\cup\delta^+(s)}} \partial^2_{va}\ell_s\ \delta\mathtt{S}_a + \sum_{\substack{d\in\delta^-(v),\\ a\in\delta^+(d)}} (\partial_d f^{\mathtt{T}}\, \partial^2_{av} d)\ \delta\mathtt{S}_a + \\ &- \left(\begin{cases} \delta[\partial_v f] & \delta^+(v) \neq \emptyset \\ -\partial_v f & \text{otherwise} \end{cases} \right) + \sum_{d\in\delta^-(v)} (\partial_v d)^{\mathtt{T}} \delta[\partial_d f]. \end{aligned} \quad (9)$$

Similarly, expanding the top block in (6) using the defini-

[1] We use $\tilde{O}(.)$ to hide factors linear in $|E| + |V|$.

tions in (3) & (6), we obtain $\forall v$,

$$\begin{aligned} &-\partial_v \mathcal{L} = \\ &\sum_{\substack{s\in v\cup\delta^{\text{-}}(v),\\ a\in v\cup\delta^{+}(s)}} \partial^2_{va}\ell_s\ \delta\mathtt{S}_a + \sum_{\substack{d\in\delta^{\text{-}}(v),\\ a\in\delta^{+}(d)}} (\lambda_d^{\mathtt{T}}\,\partial^2_{av}d)\ \delta\mathtt{S}_a + \\ &-\left(\begin{cases}\delta\lambda_v & \delta^+(v)\neq\emptyset\\ 0 & \text{otherwise}\end{cases}\right) + \sum_{d\in\delta^{\text{-}}(v)} (\partial_v d)^{\mathtt{T}}\ \delta\lambda_d, \end{aligned} \tag{10}$$

where,

$$\partial_v\mathcal{L} = \begin{cases} \sum_{s\in v\cup\delta^{\text{-}}(v)}\partial_v\ell_s + \sum_{d\in\delta^{\text{-}}(v)}\lambda_d\cdot\partial_v d, & \delta^+(v)=\emptyset \\ \sum_{s\in v\cup\delta^{\text{-}}(v)}\partial_v\ell_s + \sum_{d\in\delta^{\text{-}}(v)}\lambda_d\cdot\partial_v d & \text{otherwise} \\ -\lambda_v \end{cases} \tag{11}$$

The result follows from equations (7), (9), (10) & (11).

□

GRAPHICAL NEWTON: The above theorem immediately yields the following optimization algorithm,

Algorithm 1 Graphical Newton

1: **Input:** initial $\mathtt{S}_X$, tuple $(\mathcal{G}, \{\varphi_v\}, \{\ell_v\})$
2: **repeat**
3: Compute $f, \{\partial_v f\}, \{\partial^2\varphi_v\}$ from (2), (7).
4: Compute the SQP step from (6), with $\lambda_v = \partial_v f, \forall v$.
5: Compute step-length η *via* linesearch on inputs $\mathtt{S}_X$.
6: Update inputs: $\mathtt{S}_X \leftarrow \mathtt{S}_X + \eta\delta\mathtt{S}_X$.
7: **until** $||\partial_X f|| \leq \epsilon$

The run-time of every iteration in Algorithm 1 depends crucially upon the time required to solve (6). The run-time bounds for solving such KKT systems is taken up later in the paper.

3.3 Extension to equality constraints

Consider optimization problems, which have equality constraints in addition to the structured objective from before,

$$\begin{aligned} \min_{\mathtt{S}_{x_1},\ldots,\mathtt{S}_{x_n}} &\left(f \triangleq \sum_{v\in V[\mathcal{G}]} \ell_v(\mathtt{S}_{v\cup\delta^+(v)})\right), \\ \mathtt{S}_v \leftarrow \varphi_v(\mathtt{S}_{\delta^+(v)}),&\quad \forall v\in V[\mathcal{G}], \delta^+(v)\neq\emptyset, \\ c(\mathtt{S}_C) &= 0, \end{aligned} \tag{12}$$

where $c(\cdot)=0$ is an additional equality constraint, which depends on the variables $C\subset V[\mathcal{G}]$. The Lagrangian for this problem is given by,

$$\hat{\mathcal{L}}(\mathtt{S}_{V[\mathcal{G}]}, \lambda) = \mathcal{L}(\mathtt{S}_{V[\mathcal{G}]}, \lambda_{V\setminus X}) + \lambda_h^{\mathtt{T}} c(\mathtt{S}_C), \tag{13}$$

where $\mathcal{L}$ is as defined in (13), and $\lambda_{V\setminus X}$ is the corresponding set of multipliers; the variable λ, being the concatenation of λ_c and all multipliers, $\lambda_{V\setminus X}$, appearing in (13).

Theorem 1 can be applied to this problem by treating $\lambda_c^T c(\mathtt{S}_C)$ as another cost function in the objective, while also including the constraint in the KKT system (6). The iteration can then proceed by solving the KKT system with $\lambda_v = \partial_v(f + \lambda_c^T c), \forall v$, and using a merit function for the linesearch procedure; the variables $(\mathtt{S}_X, \lambda_c)$ are updated accordingly. We omit the proof for the validity of this method.

4 Message Passing

The classical run-time bound for Cholesky factorization (*i.e* Gaussian Belief Propagation [2]) [4] [18], cannot be extended to problems such as (6), because of the appearance of linear constraints. Such bounds for structured KKT systems, do not appear to be known within the sparse linear algebra community [2].

In this section, we provide a Message Passing algorithm for solving such KKT systems, and show that it has a run-time bound of $\tilde{O}(\text{tw}^3)$ [3], given the tree-decomposition.

4.1 Hypergraph structured QPs

For a hypergraph $\mathcal{H}$, denote the adjacency and incidence matrices by $\mathcal{A}[\mathcal{H}]$ & $\mathcal{B}[\mathcal{H}]$ respectively,

$$\begin{aligned} &\mathcal{A}[\mathcal{H}] \in \mathbb{R}^{|V[\mathcal{H}]|\times|V[\mathcal{H}]|}, \qquad \mathcal{B}[\mathcal{H}] \in \mathbb{R}^{|E[\mathcal{H}]|\times|V[\mathcal{H}]|}, \\ &\mathcal{A}[\mathcal{H}]_{uv} = \begin{cases}1 & \exists e\in E[\mathcal{H}],\ u,v\in e\\ 0 & \text{otherwise}\end{cases} \\ &\mathcal{B}[\mathcal{H}]_{eu} = \begin{cases}1 & u\in e\\ 0 & \text{otherwise}\end{cases} \end{aligned} \tag{14}$$

Given such a hypergraph $\mathcal{H}$, the family of QPs we're interested in solving is the following,

$$\begin{aligned} \min_x \sum_{e\in E[\mathcal{H}]} &\frac{1}{2}\mathtt{S}_e^{\mathtt{T}} Q_e \mathtt{S}_e - b_e^{\mathtt{T}}\mathtt{S}_e, \\ \forall e\in E[\mathcal{H}],&\quad G_e\mathtt{S}_e = h_e. \end{aligned} \tag{15}$$

Assuming that the QP has a bounded solution and that the constraints are full rank, the minimizer to (15) is given by the solution to the following KKT system,

$$\begin{aligned} \begin{bmatrix} Q & G^{\mathtt{T}} \\ G & 0 \end{bmatrix}\begin{bmatrix} x \\ \lambda \end{bmatrix} = \begin{bmatrix} b \\ h \end{bmatrix}, \\ x, b\in\mathbb{R}^{|V|},\quad \lambda, h\in\mathbb{R}^M, \end{aligned} \tag{16}$$

[2] Gaussian-BP, computes the LU decomposition of a matrix
[3] The tilde hides factors linear in $|V[\mathcal{H}]|, |E[\mathcal{H}]|$.

where Q, G, λ, x, b are concatenation of terms defined in (15) respectively. The sparsity/support of (16) is closely related to $\mathcal{H}$, since,

$$\begin{aligned}&\text{supp}(Q) \subseteq \text{supp}(\mathcal{A}[\mathcal{H}]),\\ &\forall i, \exists e, \text{supp}(G_{i,:}) \subseteq \text{supp}(\mathcal{B}[\mathcal{H}]_{e,:}).\end{aligned}$$

Every row of the constraint, $G_{i,:}$, has the same sparsity as some edge $e \in E[\mathcal{H}]$.

TREE DECOMPOSITION: Extending the notion of Dynamic Programming to non-trees (including Hypergraphs) requires a partitioning of the graph so as to satisfy a *lifted* notion of being a tree [11]. Tree decomposition captures the essence of such graph partitions,

Definition 1 *(Tree decomposition) A tree-decomposition of a hypergraph $\mathcal{H}$ consists of a tree $\mathcal{T}$ and a map $\chi : V[\mathcal{T}] \to 2^{V[\mathcal{H}]}$, such that,*

i (Vertex cover) $\cup_{i \in V[\mathcal{T}]} \chi(i) = V[\mathcal{H}]$.

ii (Edge cover) $\forall e \in E[\mathcal{H}],\ \exists i \in V[\mathcal{T}], e \subset \chi(i)$.

iii (Induced sub-tree) $\forall u \in V[\mathcal{H}],\ \mathcal{T}_u \triangleq \mathcal{T}[\{i \in V[\mathcal{T}] | u \in \chi(i)\}]$ *is a non-empty subtree*

The tree-width of a tree-decomposition $\mathcal{T}$ is defined to be $\text{tw}(\mathcal{T}) = \max_{v \in V[\mathcal{T}]} |\chi(v)| - 1$. *The tree-width of a graph $\mathcal{H}$ is defined to be the minimal tree-width attained by any tree-decomposition of $\mathcal{H}$.*

We define the vertex-induced subgraph in what follows to be $\mathcal{H}[S] \triangleq (V[\mathcal{H}], \{e \cap S, e \in E[\mathcal{H}]\})$. The following lemma ensures that such a decomposition ensures *local dependence* [11].

Lemma 1 *(Edge separation) Deleting the edge $xy \in E[\mathcal{T}]$, renders $\mathcal{H}[V \backslash (\chi(x) \cap \chi(y))]$ disconnected.*

HYPERTREE STRUCTURED QP: The tree-decomposition itself can be considered a Hypergraph, $(V[\mathcal{H}], \{\chi(u), \forall u \in V[\mathcal{T}]\})$. Such a *Hypertree*[4] can also be thought of as a Chordal graph [18]. We assume henceforth that the given graph $\mathcal{H}$ is a hypertree, and that $\mathcal{T}$ is its tree-decomposition.

The gather stage of the Message Passing algorithm, is illustrated in Algorithm 2. [5]

The function, Factorize, computes the partial LU decomposition of its arguments; we describe below, its operation. Denote the vertices that are interior to l by $\iota = \chi(l) \cap \chi(p)$,

[4]There are multiple definitions of a *Hypertree*; we use the term to mean a maximal Hypergraph, whose tree-decomposition can be expressed in terms of its edges.

[5]Note that the addition is performed vertex label-wise in Line 6 of Algorithm 2.

Algorithm 2 Graphical QP

1: **Given**: $\mathcal{T}, \mathcal{H}, \{Q_e\}, \{b_e\}, \{G_e\}, \{h_e\}$.
2:
3: **function** GatherMessage$(l, p, \mathcal{T})$
4: $(\tilde{Q}_l, \tilde{b}_l, \tilde{G}_l, \tilde{h}_l) \leftarrow (Q_l, b_l, G_l, h_l)$
5: **for** $c \in \delta_{\mathcal{T}}(l) \backslash p$ **do**
6: $(Q_{c\to l}, G_{c\to l}, b_{c\to l}, h_{c\to l}) \leftarrow$ GatherMessage$(c, p, \mathcal{T})$
7: $(\tilde{Q}_l, \tilde{b}_l) \leftarrow (\tilde{Q}_l, \tilde{b}_l) + (Q_{c\to l}, b_{c\to l})$
8: $\tilde{G}_l \leftarrow [\tilde{G}_l; G_{c\to l}], \tilde{h}_l \leftarrow [\tilde{h}_l; h_{c\to l}]$
9: **end for**
10: **return** Factorize$(\chi(l), \chi(p), \tilde{Q}_l, \tilde{b}_l, \tilde{G}_l, \tilde{h}_l)$
11:
12: **function** Factorize$(\chi(l), \chi(p), \tilde{Q}, \tilde{b}, \tilde{G}, \tilde{h})$
13: $(\xi, \iota) \leftarrow (\chi(l) \backslash \chi(p), \chi(l) \cap \chi(p))$
14: $r \leftarrow \text{rank}(\tilde{Q}_{\iota,\iota})$
15: **return** Gaussian-BP messages from (17).
16: **return**

and those on the boundary (*i.e* common to p, l) by $\xi = \chi(l) \backslash \chi(p)$, and let $r = \text{rank}(\tilde{Q}_{\iota,\iota})$. The function computes Gaussian-BP messages from block pivots $\{2, 3\}$ to $\{1, 4\}$ in (17). Note that, unlike Gaussian-BP, the matrices in (17) are not necessarily positive definite, but are however invertible.

$$\begin{bmatrix} \tilde{Q}_{\xi\xi} & \tilde{Q}_{\iota\xi}^{\mathsf{T}} & \tilde{G}_{:r,\xi}^{\mathsf{T}} & \tilde{G}_{r:,\iota}^{\mathsf{T}} \\ \tilde{Q}_{\iota\xi} & \tilde{Q}_{\iota\iota} & \tilde{G}_{:r,\iota}^{\mathsf{T}} & \tilde{G}_{r:,\iota}^{\mathsf{T}} \\ \tilde{G}_{:r,\xi} & \tilde{G}_{:r,\iota} & 0 & 0 \\ \tilde{G}_{r:,\xi} & \tilde{G}_{r:,\iota} & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathtt{s}_\xi \\ \mathtt{s}_\iota \\ \lambda_{:r} \\ \lambda_{r:} \end{bmatrix} = \begin{bmatrix} \tilde{b}_\xi \\ \tilde{b}_\iota \\ \tilde{h}_{:r} \\ \tilde{h}_{r:} \end{bmatrix} \tag{17}$$

Gaussian Belief-Propagation is essentially a restatement of LU decomposition [4]. Gaussian-BP consists of messages of the form [21] [18],

$$\begin{aligned}\mu_{i\to j} := [J_{i\to j}, h_{i\to j}] &= [J_{ii}, h_i] - \sum_{k \in \delta(i) \backslash j} J_{ik} J_{k\to i}^{-1} [J_{ki}, h_{k\to i}],\\ \mu_i &= J_{i\to j}^{-1}(h_{i\to j} - J_{ij}\mu_j),\end{aligned} \tag{18}$$

where $J\mu = h$ is the equation that is to be solved. These can be replaced by appropriate square-root forms to obtain instead, an LDL decomposition.

Theorem 2 *The linear equation (16) can be solved in time $\tilde{O}(\text{tw}(\mathcal{H})^3)$, given the minimal tree-decomposition* via *Algorithm 2.*

Proof. The correctness of the algorithm follows from Lemma 1. The bound holds trivially if, $\text{rank}\, \tilde{G} \leq \text{rank}\, \tilde{Q}_{\iota,\iota}$, at every step of the algorithm. Otherwise, by realizing that $\tilde{G}_{l\to p}$, can't have rank more than $|\chi(p)|$, the proof follows. □

It follows from Theorem 2, that the KKT system in Algorithm 1 can be solved in time $\tilde{O}(\mathrm{tw}(\hat{\mathcal{G}})^3)$, where $\hat{\mathcal{G}}$ is the moralization of the computational graph $\mathcal{G}$.

The above proof also ensures that the equivalent sparse LU/LDL decomposition [4], with the same pivot order, also has the same run-time. Since decompositions of indefinite systems are subject to instability, use of specialized solvers is generally preferable.

5 Numerical Experiments

In this section, we present preliminary numerical results with an implementation of Algorithm 1, using the MA57 solver [7]. For ensuring convergence in constrained problems, an augmented Lagragian merit function was used [8]. The implementation was tested on the following non-standard control problems.

SPRING-DAMPER LIMIT CYCLE: Consider the following spring-damper limit-cycle problem [14],

$$\begin{aligned} &\min_{x_0, u[0,T]} \int \ell(x, \dot{x}, u) \mathtt{d}t, \\ &\ddot{x} = -(x^3 + \dot{x}^3)/6 + u, \\ &x(0) = x(T) = x_0, \dot{x}(0) = \dot{x}(T) = \dot{x}_0, \end{aligned} \tag{19}$$

where,

$$\ell(x, \dot{x}, u) = (1 - \mathrm{e}^{(\dot{x}_i - 2)^2} - \mathrm{e}^{-(\dot{x}_i + 2)^2}) + \frac{1}{2}||u_i||_2^2.$$

Discretising the derivatives by finite differences, $\dot{x} \approx \Delta x_i / \Delta t = (x_i - x_{i-1})/\Delta t$, this can be written as the following structured optimization problem,

$$\begin{aligned} &\min_{x_0, \{u_i\}_0^m} \sum_1^N \ell(x_i, \Delta x_i/\Delta t, u_i), \\ &x_{i+1} \leftarrow x_i + \Delta x_i + (\Delta t)^2 \left[-(x^3 + (\Delta x_i/\Delta t)^3)/6 + u_i\right]. \\ &x_0 = x_{N-2}, x_1 = x_{N-1}. \end{aligned} \tag{20}$$

For $N = 100, \Delta t = 0.1$, with random initializations, the problem showed robust convergence; often taking no more than ten SQP iterations. The optimal limit cycle, and the convergence curves for one run of the algorithm are shown in (Figure 4.1).

6 Discussion

We have shown that the Newton step can be computed in time $\tilde{O}(\mathrm{tw}^3)$, where 'tw' is the tree-width of the computational graph. We have also derived extensions to constrained problems, and provided numerical examples. The technique presented herein, also generalizes many specialized algorithms in control.

In certain control problems, the solution to the KKT system, itself can be written in *feedback form*. Given a LU decomposition of the KKT system, one can replace the back-substitution phase by U, with a function evaluation that uses L as a *control feedback* [10]. It is unclear if such techniques can be generalized, and whether they can be made independent of the pivot-order used for solving the system.

A competing method for exploiting the structure of objectives such as (3), is by the use Hessian vector product AD routines in conjugation with CG-like methods. Computing the Hessian vector product takes time $\tilde{O}(\omega(\hat{\mathcal{G}})^2)$, where $\hat{\mathcal{G}}$ is a moralization of the computational graph [6]. By contrast, if the computational graph were chordal, then computing the Newton-step *via* Algorithm 1 is only $\tilde{O}(\omega(\hat{\mathcal{G}})^3)$. The latter is more economical when the cliques of a graph are small in comparison to the order of the graph. The ill-conditioned nature of structured objectives may also lead to bad convergence properties for CG algorithms.

For problems whose tree-widths are large, the iterative method is obviously more viable. However, following the rapid advances in approximate inference in the past two decades [18], we hope that the explicit algebraic connection to graphical models made in this paper, can be exploited in coming up with less-agnostic iterative methods.

References

[1] B. M. Bell, J. V. Burke, and G. Pillonetto. An inequality constrained nonlinear Kalman-Bucy smoother by interior point likelihood maximization. *Automatica*, 45(1):25–33, 2009.

[2] R. Bridson. An ordering method for the direct solution of saddle-point matrices. *Preprint*, 2007.

[3] T. F. Coleman and J. J. More. Estimation of sparse Hessian matrices and graph coloring problems. *Mathematical Programming*, 28(3):243–270, 1984.

[4] T. A. Davis. *Direct Methods for Sparse Linear Systems*, volume 2. SIAM, 2006.

[5] J. De O. Pantoja. Differential Dynamic Programming and Newton's method. *International Journal of Control*, 47(5):1539–1553, 1988.

[6] L. Dixon. Automatic Differentiation: Calculation of Newton Steps. In *Encyclopedia of Optimization*, pages 137–142. Springer, 2009.

[7] I. S. Duff. MA57—a code for the solution of sparse symmetric definite and indefinite systems. *ACM Transactions on Mathematical Software (TOMS)*, 30(2):118–144, 2004.

[8] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. Some theoretical properties of an augmented

lagrangian merit function. Technical report, DTIC Document, 1986.

[9] A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, 2008.

[10] D. H. Jacobson and D. Q. Mayne. *Differential Dynamic Programming*. North-Holland, 1970.

[11] J. Kleinberg and É. Tardos. *Algorithm Design*. Pearson, Addison-Wesley, 2006.

[12] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Science & Business Media, 2006.

[13] D. Ralph. A parallel method for unconstrained discrete-time optimal control problems. *SIAM Journal on Optimization*, 6(2):488–512, 1996.

[14] Y. Tassa, T. Erez, and E. Todorov. Optimal limit-cycle control recast as Bayesian inference. In *Proceedings of thh IFAC world congress*. Citeseer, 2011.

[15] E. Todorov and W. Li. A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *American Control Conference, 2005. Proceedings of the 2005*, pages 300–306. IEEE, 2005.

[16] M. Toussaint and C. Goerick. A Bayesian view on motor control and planning. In *From Motor Learning to Interaction Learning in Robots*, pages 227–252. Springer, 2010.

[17] J. Utke. Efficient Newton steps without Jacobians. In M. Berz, C. H. Bischof, G. F. Corliss, and A. Griewank, editors, *Computational Differentiation: Techniques, Applications, and Tools*, pages 253–264. SIAM, Philadelphia, PA, 1996.

[18] M. J. Wainwright and M. I. Jordan. Graphical Models, Exponential Families, and Variational Inference. *Foundations and Trends® in Machine Learning*, 1(1-2):1–305, 2008.

[19] S. J. Wright. Solution of discrete-time optimal control problems on parallel computers. *Parallel Computing*, 16(2):221–237, 1990.

[20] S. J. Wright. Interior point methods for optimal control of discrete time systems. *Journal of Optimization Theory and Applications*, 77(1):161–187, 1993.

[21] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Generalized Belief Propagation. In *NIPS*, volume 13, pages 689–695, 2000.

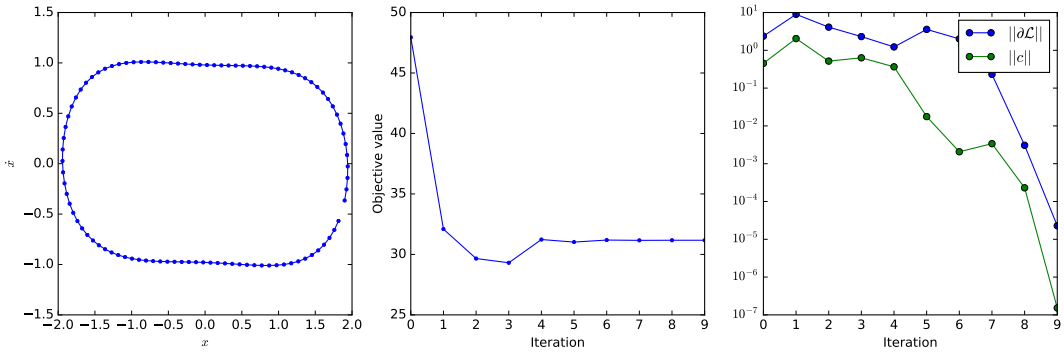

Figure 2: LEFT: An optimal limit cycle for the system $\ddot{x} = -(x^3 + \dot{x}^3)/6 + u$. MIDDLE: Convergence of the objective function for the limit-cycle problem (20). RIGHT: Convergence in norm, of the Lagrangian gradient, and constraint deviation.